Université de Montréal
École des Hautes Études Commerciales
École Polytechnique

Université de Montréal
C.P. 6128, Succursale Centre-ville,
Montréal (Québec) Canada H3C 3J7

Téléphone +1-514-343-7575
Télécopieur (Fax) +1-514-343-7121
crt@crt.umontreal.ca
www.crt.umontreal.ca/CRT/

**CRT-2000-42**

**Centre de recherche sur les transports (C.R.T.)**
*Centre for Research on Transportation*

# *DIVERSIFICATION STRATEGIES IN LOCAL SEARCH FOR A NONBIFURCATED NETWORK LOADING PROBLEM*

**by**

**Bernard Gendron**
**Jean-Yves Potvin**
**Patrick Soriano**

**December 2000**

**PUBLICATION**
**CRT-2000-42**

# *DIVERSIFICATION STRATEGIES IN LOCAL SEARCH FOR A NONBIFURCATED NETWORK LOADING PROBLEM**

**by**

**Bernard Gendron**[1]
**Jean-Yves Potvin**[1]
**Patrick Soriano**[2]

* This research was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC), and by the Fonds pour la Formation de Chercheurs et l'Aide à la Recherche (FCAR).

1 Département d'informatique et de recherche opérationnelle et Centre de recherche sur les transports, Université de Montréal, C.P. 6128, succursale Centre-ville, Montréal, Canada H3C 3J7

2 École des Hautes Études Commerciales et Centre de recherche sur les transports, Université de Montréal, C.P. 6128, succursale Centre-ville, Montréal, Canada H3C 3J7

Dépôt légal – Bibliothèque nationale du Québec
Bibliothèque nationale du Canada, 2000

102-03121/

**Abstract**

This paper examines a variant of the network loading problem, a network design problem that arises in the telecommunications industry. In this problem, we have to install facilities of fixed capacity on the edges of an undirected network to carry the flow from a central vertex to a set of demand vertices so that the total facility installation cost is minimized. To satisfy the demand at any vertex, the flow must follow a single path: this is the nonbifurcated version of the problem. We propose heuristics based on alternating between local search and construction steps. At each construction step, except the initial one, we perform a diversification strategy to provide a new initial point to the local search procedure. We experimentally compare different diversification strategies, using large-scale instances with up to 500 vertices.

**Key words :** network loading problem, metaheuristics, local search, diversification.

**Résumé**

Dans cet article, nous étudions une variante du problème de chargement de réseau, un problème de conception de réseau dans le secteur des télécommunications. Ce problème consiste à installer des équipements de capacité fixe sur les arêtes d'un réseau non orienté afin de satisfaire les demandes d'un ensemble de sommets à partir d'un sommet central. Pour établir la connection entre le sommet central et chaque sommet de demande, on ne peut utiliser qu'un seul chemin. Nous proposons des heuristiques basées sur l'alternance entre des phases de recherche locale et de construction. À chaque phase de construction, sauf la première, nous utilisons une stratégie de diversification pour générer un nouveau point de départ pour la recherche locale. Afin de comparer différentes approches de diversification, nous présentons des résultats numériques sur des problèmes de grande taille ayant jusqu'à 500 sommets.

**Mots-clés :** problème de chargement de réseau, métaheuristiques, recherche locale, diversification.

# 1 Introduction

Network design models find applications in various fields such as computer networks, transportation, manufacturing and telecommunications [1, 10, 20, 21]. In this paper, we study a variant of the network loading problem [19] where we have to install facilities of fixed capacity on the edges of an undirected network to carry the flow from a central vertex to a set of demand vertices so that the total facility installation cost is minimized. To satisfy the demand at any vertex, the flow must follow a single path: this is the nonbifurcated version of the problem. We assume there is a fixed and limited number of facility types (the typical application we consider has nine facility types). Each facility type has a capacity and a cost of installation. The cost increases with the capacity but exhibits substantial economies of scale. For example, if we have two types of facilities, the first one with capacity 1 and the second one with capacity $C > 1$, the cost of installing one facility of the second type is significantly less than $C$ times the cost of installing one facility of the first type. This problem is known to be NP-hard, even if we consider some of its simplest special cases [18].

The literature on similar problems has mainly focused on exact approaches based on mathematical programming techniques (cutting planes, Lagrangean relaxation) [4, 8, 9, 12, 17, 18, 19, 22]. These approaches can only solve instances of relatively small dimensions (typically, networks with less than 50 vertices). Heuristic methods are thus particularly indicated to address large-scale instances of realistic sizes.

In [3], we propose a tabu search heuristic to obtain effective approximate solutions to this problem. The so-called 1-opt neighborhood structure used in this algorithm is based on replacing one path associated to one demand vertex by an alternative path that allows to carry the flow to the same demand vertex. To provide new starting points for the search, this heuristic also includes a long term adaptive memory [24]. Although the method outperforms a descent heuristic based on a 2-opt neighborhood (replace two paths by two alternative ones), it has not been compared to other, more sophisticated, approaches. In particular, the proposed adaptive memory approach has not been measured against other diversification strategies. The objective of the present paper is twofold: (1) to propose other diversification approaches that provide new starting points for the local search; (2) to compare their performances on large-scale telecommunications instances with up to 500 vertices. We also investigate the behavior of the proposed diversification strategies when we use a simple 1-opt descent heuristic instead of the local search method based on tabu mechanisms.

The paper is organized as follows. Section 2 presents a mathematical formulation of the problem. In Section 3, we describe the framework that forms the basis for our implementation of the local search (i.e., descent or tabu search) and diversification procedures. The diversification strategies are presented in Section 4. In Section 5, we report computational results on typical telecommunications data.

## 2 Problem Formulation

Let $G = (V, E)$ be an undirected graph where $V$ is the vertex set of cardinality $n$, $E$ is the edge set and $D \subseteq V$ is the subset of demand vertices of cardinality $r$. We further introduce the following notation:

$T$ is the set of facility types,
$P_i$ is the set of paths from the central vertex to demand vertex $i$,
$d_i$ is the demand at vertex $i$,
$c_t$ is the cost of a facility of type $t$ (per unit of length),
$w_t$ is the capacity of a facility of type $t$,
$l_e$ is the length of edge $e$,
$u_e$ is the initial capacity on edge $e$,
$\delta_{ej}$ is equal to 1 if edge $e$ is on path $j$, 0 otherwise.

We also introduce two sets of variables: the variables $x_{ij}$ which are equal to 1 if path $j \in P_i$ is used to service demand vertex $i$, 0 otherwise; and the design variables $y_{te}$ which are equal to the number of facilities of type $t$ to be installed on edge $e$. The problem we address can then be formulated as follows:

Minimize

$$\sum_{e \in E} \sum_{t \in T} l_e c_t y_{te}$$

subject to

$$\sum_{i \in D} \left( \sum_{j \in P_i} x_{ij} \delta_{ej} \right) d_i \leq u_e + \sum_{t \in T} w_t y_{te}, \ \forall e \in E,$$

$$\sum_{j \in P_i} x_{ij} = 1, \ \forall i \in D,$$

$$x_{ij} \in \{0, 1\}, \ \forall i \in D, \ \forall j \in P_i,$$

$$y_{te} \text{ integer}, \forall t \in T, \ \forall e \in E.$$

In this model, the first set of constraints specifies that the capacity installed on the network must be sufficient to handle the flow on each edge. The second set of constraints requires nonbifurcated flows, that is, the use of a single path from the central vertex to each demand vertex. The objective is to minimize the installation costs of any additional capacity on the edges of the network to satisfy the demand at each vertex.

## 3 Local Search Heuristics

The heuristics we propose for finding good approximate solutions to the problem are based on alternating between construction steps and local search phases. Each construction step consists in identifying a feasible solution that serves as a new starting point to the local search procedure. At the start of the algorithm, a constructive heuristic provides an initial feasible solution, while at subsequent construction steps, we use a diversification approach that might exploit informations gathered along the history of

the search process. The local search procedure is an iterative neighborhood search strategy. We implement either one of the two following methods: a pure descent approach that moves to a neighboring solution only if this solution improves over the best solution found during this local search phase, or a tabu search (TS) procedure that moves to an improving solution in the neighborhood, when one can be found, but also allows moves that degrade the objective function. To avoid cycling when using TS, a short term memory, known as the tabu list, stores previously visited solutions or components of previously visited solutions. It is then forbidden or tabu to come back to these solutions for a certain number of iterations. In the remainder of the paper, we assume familiarity with the principles of tabu search; for further details, the interested reader is referred to [11].

We first briefly sketch the general algorithmic structure of our heuristics. We then present the neighborhood used in the local search steps, while the components of the construction steps (initialization and diversification) are described in Section 4.

1. while *stopping criterion of the main loop* is not met do:

   **construction step**

   a. if we are at the start of the algorithm then

      generate an initial solution $s$ by a constructive heuristic;

      $s^{best} = s$;

      otherwise

      generate $s$ by a diversification procedure;

   **local search step**

   b. $s^* = s$;

   c. while *stopping criterion of the local search loop* is not met do:

      generate a neighborhood of $s$ and select the best solution $s'$ in the neighborhood of $s$;

      store informations about $s'$ for use in the diversification procedure, if indicated (see Section 4);

      if $s'$ is better than $s^*$ then $s^* = s'$;

      $s = s'$;

   d. if $s^*$ is better than $s^{best}$ then $s^{best} = s^*$;

2. output $s^{best}$

In this algorithm, we use the variables $s^*$ and $s^{best}$ to represent the best solution of the current local search phase and the best overall solution, respectively. The algorithm terminates when the total CPU time spent in the main loop exceeds some value $time_{max}$. The stopping criterion of the local search loop depends on the method, descent or TS. When descent is used, the stopping rule is the usual one: $s$ has not improved over $s^*$. In the case of TS, we stop the search when the number of consecutive iterations without improving $s^*$ has reached a user-supplied value $iter^*$.

### 3.1 Neighborhood

A solution in the neighborhood of any feasible solution is obtained by replacing, for a particular demand vertex, the current path used to reach that vertex by an alternative one. To determine which of these neighboring solutions appears the most promising, we need to compute the cost incurred by the replacement of one path for a particular demand vertex $i$. To that end, we have to compute the edge costs incurred to satisfy the demand of vertex $i$, given that the other $r-1$ demand vertices are connected to the central vertex using the paths in the current solution. This can be easily done because, on every edge, we know the exact amount of additional capacity needed to satisfy the demand of vertex $i$. Since we have a fixed and limited number of facility types, it is easy to determine *a priori*, before the start of the algorithm, which facilities need to be installed for any flow between 0 and $\sum_{i=1}^{r} d_i$ (see subsection 5.1). This information is used to compute the edge costs incurred to satisfy the demand of vertex $i$.

We use these edge costs to find the *best* alternative path for each demand vertex $i$. We first compute the shortest path from the central vertex to $i$. If this path is different than the current path, it corresponds to the best alternative path. Otherwise, we have to compute the second shortest *elementary*, or loopless, path. While finding the shortest path by Dijkstra's algorithm [5] is an easy task, computing the second shortest elementary path is harder. There exists polynomial-time exact algorithms to solve this problem [13]. However, a heuristic approach based on a direct adaptation of Dijkstra's algorithm is indicated in our context since it is particularly efficient on sparse networks [25], which are typical in telecommunications.

In this adaptation, the scalar label associated with every vertex $i$ is replaced by a vector of size 2 whose $p$-th element is an upper bound on the length of the $p$-th shortest path to vertex $i$. The method proceeds similarly as in Dijkstra's algorithm, by selecting first the vertex $i$ with the smallest label and then, by updating the labels of the vertices adjacent to $i$. This basic label setting strategy is modified in a simple way to forbid paths with cycles: once the minimum temporary label is chosen, any adjacent vertex to $i$ already identified on the tentative path, is excluded from further consideration. Clearly, this approach always generates elementary paths. However, it does not necessarily produce the second shortest one, as some paths may be overlooked, and thus, it constitutes a heuristic to the problem of finding the second shortest elementary path. The procedure is described in greater detail in [3].

In spite of the efficiency of the method we use for computing the best alternative path, exploring the whole 1-opt neighborhood at each iteration of the local search procedure remains computationally elusive. In particular, the shortest paths must be computed independently for each demand vertex since the costs must be reevaluated when a new vertex is considered. Therefore, instead of evaluating the best 1-opt move for each of the $r$ demand vertices, we allow to compute the best alternative path for only a fraction $pct_1$ of the $r$ demand vertices, where $pct_1$ is a user-supplied parameter.

Both descent and TS share the same neighborhood structure. In the case of TS, however, we use the tabu list mechanism to further restrict the selection of the demand vertices for which we evaluate a 1-opt move. At every iteration of TS, once we identify

the best solution in the neighborhood, the associated demand vertex is declared tabu for a number of iterations randomly chosen in the interval $[tabu_{min}, tabu_{max}]$. Consequently, the path leading to that vertex cannot be changed for that number of iterations. By associating a tabu status to the vertices rather than to the paths themselves, we favor a more restrictive approach, as many moves may be forbidden to reduce the risks of cycling. However, we might still apply a move, even if it is tabu, if it leads to a better solution than the best one obtained thus far (this is the usual aspiration criterion).

# 4 Diversification Strategies

During the construction phase of our heuristics, we build a new solution by using one of the following two approaches:

- *Sequence independent construction (SI)*: We obtain a solution by identifying one path from the central vertex to each demand vertex, independently of each other.
- *Sequence dependent construction (SD)*: We build a solution by examining the demand vertices in some order. For each demand vertex, we compute the shortest path using the edge costs incurred to satisfy the demand of that vertex, given that some of the demand vertices are already connected to the central vertex.

The initialization and adaptive memory approaches implemented in [3] are examples of SI procedures. There, the initialization consists in selecting the shortest paths, with respect to the edge lengths, from the central vertex to each demand vertex. In the case of the adaptive memory, we use $r$ fixed size compartments, each one containing paths leading from the central vertex to an associated demand vertex. Each path has a score which corresponds to the objective value of the best visited solution that used that path. When a new current solution is obtained by the local search, the paths in this solution are stored in their associated compartment if: (1) the compartment is not full or (2) the path is better than the worst path in the compartment, in which case the new path takes its place. The construction step consists in randomly selecting a path from each compartment, where the selection is biased towards the paths with the best scores (we describe in Appendix A the general procedure we use to perform the biased selection of one element from a list of ordered elements, a procedure that is also used in other diversification approaches).

A possible implementation of an SD procedure would proceed as follows. Initially, we determine, for each demand vertex, the shortest path from the central vertex, by using, for each edge, the cost of the facilities required to satisfy the corresponding demand. We then select among these shortest paths the one with minimum cost per unit of demand. At each subsequent step, we proceed in a similar way by computing the shortest path for each demand vertex that is not yet connected to the central vertex, and by selecting the path with minimum per unit cost. It is important to note that the edge costs are adjusted at each iteration of this procedure, since the paths selected at the previous iterations determine the facilities to install at the current iteration. This greedy procedure can

be slightly modified to provide several different starting points. For example, instead of selecting the path with minimum per unit cost, we could select one at random from a list of paths that are not "too far" from the best one (one way to implement this idea is by using the mechanism proposed in greedy randomized adaptive search procedures, or GRASP [6, 7, 23]).

This approach suffers from a serious drawback: at each iteration, it requires the computation of the shortest path for each demand vertex not yet connected to the central vertex. Thus, a total of $r(r+1)/2$ shortest path computations are necessary before we obtain a solution. If we could compute only the shortest path for one demand vertex at each iteration, the computational effort would be significantly reduced. To do so, we can impose an *a priori* order for the selection of the next demand vertex for which to compute the shortest path from the central vertex. For example, if we represent the demand vertices in decreasing order of their demand values, we would then select in priority the vertex with the greatest demand and compute its shortest path from the central vertex. This order of selection is motivated by the structure of the costs, which suggests that substantial economies of scale can be achieved if we first install higher capacity facilities (this way, we can expect the smaller demand vertices to be connected for "free"). In general, this approach provides significantly better initial solutions than the SI procedure used in [3] and described above. This is easily explained since, even if the SI procedure finds the best path to reach each vertex individually, the overall solution can still be far from optimality, as these paths typically share common edges. Consequently, in all experiments presented in Section 5, we have obtained an initial solution by using an SD procedure based on ordering the vertices according to their demand magnitudes. The superiority of this SD procedure also suggests that, at each diversification step, a similar approach could be used in place of the adaptive memory procedure described above.

For example, instead of using a deterministic *a priori* order, we can choose at random the next demand vertex for which to compute its shortest path. To retain the effectiveness of the approach, however, the random selection must be biased towards the vertices with the greater demand values, according to the procedure described in Appendix A. Unfortunately, this *Greedy multistart* procedure also suffers from a serious drawback: it is a pure multistart approach, and as such, it does not take into account the history of the search process, unlike the adaptive memory procedure.

To alleviate this problem, we can at least utilize the recent history of the search. For example, we can retain some particularities of the best solution $s^*$ found during the last local search step, thus forming a partial solution, and then construct a complete feasible solution by using an SD procedure. If we want to generate a new solution that is close to $s^*$, we can perform a 2-opt move, which consists in replacing two paths associated to two demand vertices $i$ and $h$ by two alternative ones. More precisely, a 2-opt move is implemented as follows: (1) find the best alternative path to reach vertex $i$, then the best alternative path to reach vertex $h$; (2) repeat step (1) with $i$ and $h$ interchanged; (3) the two best alternative paths correspond to the best of the ones found in (1) and (2). We could select the best move in the entire 2-opt neighborhood, formed by all possible 2-opt moves. However, since we perform these moves to achieve

diversification, we will select the best move in only a small fraction $pct_2$ of the whole 2-opt neighborhood, where $pct_2$ is a user-supplied parameter. To this end, we compute two fractions $p_1$ and $p_2$ such that $p_1 \times p_2 = pct_2$, according to the recursive formula $p_i = \max\{1/r, pct_2/p_{i+1}\}$, where $p_3 = 1$. We then choose randomly $p_1 \times r$ demand vertices, and for each of them, perform a random selection among the other demand vertices according to the fraction $p_2$. Because performing only one 2-opt move might be insufficient to achieve diversification, we allow to perform a fixed (usually small) number $iter_2$ of 2-opt moves. Note that each move is performed even if it deteriorates the current solution. This *2-opt neighborhood* diversification strategy, combined with the 1-opt local search procedure, is reminiscent of the variable neighborhood search approach [14, 15, 16]. We could extend it to include similar $k$-opt moves with $k \geq 3$. However, preliminary experiments on the 3-opt neighborhood have revealed that the evaluation of even a very small fraction of that neighborhood is too time-consuming to justify its potential benefits.

The idea of using general $k$-opt moves in the diversification step is nonetheless interesting, especially if only one move is evaluated. This can be achieved as follows: remove $k$ paths from the current best solution $s^*$, thus obtaining a partial solution, and complete this solution by using an SD procedure, where the next demand vertex to consider is chosen at random, with a bias towards the vertices with the greater demand values. At each diversification step, the number $k$ of paths to replace is chosen in an interval $[\lceil pct_k^1 \times r \rceil, \lceil pct_k^2 \times r \rceil]$, where $pct_k^1$ and $pct_k^2$ are parameters. Then, the $k$ demand vertices are chosen at random. Note that this *Greedy k-opt move* strategy, unlike the 2-opt method described above, makes no attempt at forbidding the immediate regeneration of the same solution $s^*$. We could easily modify it accordingly, but this is not necessary since local search and diversification steps only evaluate a fraction of all possible moves. Thus, even if the same solution is visited twice, the trajectories subsequently used in each case might be significantly different.

Unlike the 2-opt neighborhood approach, the Greedy $k$-opt move procedure involves the evaluation of only one solution at each diversification step. One way to improve its performance is to generate a fixed (relatively small) number of new solutions from $s^*$ and to restart from the best of these. We can even go one step further: we can store the new, best, solutions identified in this way in a pool that also contains solutions obtained in the previous diversification steps. We can then use as starting point the best solution in this pool. Since this solution is not necessarily derived from the current $s^*$, this approach has the potential to take into account the whole history of the research process, not only the recent one (note that such a pool of elite solutions can also be seen as an adaptive memory [26]). In the remainder, we assume that the number of new solutions generated from $s^*$ and the maximum size of the pool are equal and set to *size*, a user-supplied parameter. This need not be the case in general, but this simplification has little impact, if any, on the performance of the approach. The pool of elite solutions is managed as follows. Initially, the pool is empty. At the first diversification step, *size* new solutions are inserted in the pool, and the best one is selected and removed from it. At subsequent steps, the *size* new solutions are compared to the $size - 1$ solutions remaining in the pool. The *size* best solutions among these are kept in the pool, from which we select and

remove the best one. Note that the strategy which consists in generating only one new solution from $s^*$ is a particular case of this more general approach, obtained by using $size = 1$.

To summarize, we have proposed four diversification strategies:

- *Adaptive memory (AM)*: This is the approach proposed in [3], where we obtain a new starting point by performing an SI procedure. It is based on selecting one path for each demand vertex according to the "quality" of the best solution (previously identified during the search) where this path appeared.

- *Greedy multistart (GM)*: This makes use of an SD procedure, where the random selection of the next demand vertex for which to compute the shortest path is biased towards the greater demand magnitudes.

- *2-opt neihborhood (2N)*: This approach performs a fixed, small number, of iterations. At each iteration, we select the best candidate in a fraction of the 2-opt neighborhood of the solution identified in the previous iteration. The initial solution is the best one found during the last local search phase. The new starting point is the final solution identified at the last iteration.

- *Greedy k-opt exchange (Gk)*: In this approach, we generate solutions from the best one found during the last local search phase. We store these new solutions in a fixed size pool that also contains solutions obtained in the previous diversification steps. To generate these new solutions, we evaluate a small number of $k$-opt moves, with $k$ randomly chosen. Within each move, the evaluation is performed through the same greedy SD procedure used in the GM strategy.

# 5 Computational Results

In order to evaluate the diversification strategies, we performed computational experiments on randomly generated networks with characteristics frequently observed in telcommunications applications. Before presenting the numerical results, we first introduce the cost data and the network topologies used in the experiments.

## 5.1 Cost Data

The installation costs for different types of facilities come from a real-world application. Since these costs are confidential, they are shown in generic form in Table 1. As observed in this table, it is not always optimal to use the smallest single facility that can accommodate the flow. For a flow of 100, for example, it is better to use facilities 7 and 1 with a total capacity of 102 and a total cost of 3.58, rather than facility 8 with capacity 144 and a cost of 4.37. This additional complication can be alleviated through the following observations:

| facility type | capacity | cost (per unit of length) |
|---|---|---|
| 1 | 6 | 0.55 |
| 2 | 12 | 0.73 |
| 3 | 24 | 1.03 |
| 4 | 36 | 1.39 |
| 5 | 48 | 1.67 |
| 6 | 72 | 2.31 |
| 7 | 96 | 3.03 |
| 8 | 144 | 4.37 |
| 9 | 216 | 6.33 |

Table 1: Facility types

- except for facility 9, it is never advantageous to use the same facility twice (for example, installing facility 1 twice provides 12 units of capacity, while the same capacity is obtained at lower cost using facility 2);
- facility 9 is mandatory for flows of 216 or more, as it is less expensive than any other combination of smaller facilities.

The special structure of these costs can thus be exploited to a priori determine the facilities to be installed for any flow between 0 and $\sum_{i=1}^{r} d_i$:

- for flows over 216, facility 9 is installed until a residual flow under 216 is obtained;
- for flows between 1 and 215, all possible combinations of *different* facilities are considered (except facility 9), and the combination of lowest cost is kept.

## 5.2 Randomly Generated Networks

We randomly generated general undirected networks with $n = 200$ and 500 vertices to test our algorithms. These networks are sparse and quasi-planar, like those found in practice, with edge densities (i.e., fraction of all possible edges) varying between 1% and 10%. More precisely, we generated 40 instances, 10 in each of four different types of networks:

- $n = 200$, $r = 100$ and edge density = 1%;
- $n = 200$, $r = 100$ and edge density = 10%;
- $n = 500$, $r = 250$ and edge density = 1%;
- $n = 500$, $r = 250$ and edge density = 10%.

A demand is thus associated with 50% of the vertices. The demand vertices fall into three categories: category 1 with 10 to 20 units of demand; category 2 with 40 to 60 units; category 3 with 80 to 100 units. The demand vertices are evenly distributed among the three categories. The problems come with some equipment already installed The total capacity of the network, which is the sum of the capacities over all edges of the network, is twice the total demand. This capacity is then uniformly distributed over all edges, with a random perturbation on each edge (i.e. $\pm$ 0-20%).

## 5.3 Experiments

Preliminary experiments were conducted to determine appropriate parameter values for the different heuristics. Since our objective is to compare the diversification strategies, we decided to use the same parameter values in the local search phase for all approaches. Specifically, preliminary experiments allowed us to identify the following parameter values: (1) *proportion of the 1-opt neighborhood explored at each descent iteration*: $pct_1 = 25\%$; (2) *proportion of the 1-opt neighborhood explored at each TS iteration*: $pct_1 = 10\%$; (3) *tabu list length*: $tabu_{min} = 5$, $tabu_{max} = 9$; (4) *TS stopping criterion*: maximum number of consecutive iterations without improvement, $iter^* = 10$. These values are not necessarily "optimal" for each diversification approach, but they exhibit good performances over all approaches, and thus allow to perform a fair comparison of them.

Each diversification strategy has also been calibrated. For our comparison, we use the following values: (1) size of each compartement in *Adaptive memory*: 10; (2) *2-opt neighborhood*: fraction of the neighborhood to evaluate, $pct_2 = 0.5\%$; number of 2-opt moves at each diversification step, $iter_2 = 3$; (3) number of paths, $k$, to exchange in *Greedy k-opt exchange*: $k \in [\lceil pct_k^1 \times r \rceil, \lceil pct_k^2 \times r \rceil]$, with $pct_k^1 = 5\%$ and $pct_k^2 = 20\%$. In addition, for the GM and G$k$ strategies, we have tested two values of the parameter *size*: 1 and 10 (remember that $size = 1$ corresponds to the basic variant of the diversification strategies where only one solution is generated from the best one visited during the last local seach).

Our experiments were conducted on a 64-processor, 64-GB Sun Enterprise 10000 machine, with each processor operating at 400 MHz. Because all heuritics contain stochastic features, we performed three runs of each method for each problem instance. Tables 2 and 3 compare the results obtained with the different diversification strategies for the four types of networks. For 200-vertex instances, the total CPU time is set to $time_{max}$ = 1 hour, while for 500-vertex, we increased this value to 2.5 hours. For each diversification strategy, we report three measures: $Z(s^{best})$, the value, averaged over 10 instances, of the best solution identified by the heuristic; $iter_{div}$, the average number of diversification steps; $iter_{loc}$, the average number of local search iterations performed within each diversification step.

In addition to the four diversification strategies presented in Section 4, we present the results of two other approaches: (1) a pure TS method, with no diversification ("TS"); (2) a Greedy $k$-opt move approach, with $k$ fixed to 2 ("G2"). The pure TS method

| Heuristic | density = 1% | | | density = 10% | | |
|---|---|---|---|---|---|---|
| | $Z(s^{best})$ | $iter_{div}$ | $iter_{loc}$ | $Z(s^{best})$ | $iter_{div}$ | $iter_{loc}$ |
| Initialization | 16251 | – | – | 11146 | – | – |
| TS | 15857 | 1 | 45208 | 10895 | 1 | 31219 |
| AM–descent | 15917 | 1017 | 45 | 10954 | 779 | 39 |
| AM–TS | 15918 | 1449 | 77 | 10933 | 1001 | 76 |
| GM–descent ($size = 1$) | 15941 | 3897 | 11 | 10966 | 2579 | 9 |
| GM–TS ($size = 1$) | 15949 | 2912 | 37 | 10956 | 1811 | 36 |
| GM–descent ($size = 10$) | 15941 | 1498 | 7 | 10957 | 965 | 7 |
| GM–TS ($size = 10$) | 15943 | 1340 | 28 | 10951 | 853 | 27 |
| 2N–descent | 15839 | 4753 | 4 | 10879 | 2880 | 5 |
| 2N–TS | 15880 | 12912 | 10 | 10940 | 7879 | 10 |
| G2–descent ($size = 1$) | 15870 | 47865 | 1 | 10914 | 30677 | 1 |
| G2–TS ($size = 1$) | 15855 | 13150 | 10 | 10893 | 8213 | 10 |
| G2–descent ($size = 10$) | 15876 | 24329 | 2 | 10910 | 17077 | 1 |
| G2–TS ($size = 10$) | 15870 | 10264 | 12 | 10898 | 6622 | 11 |
| G$k$–descent ($size = 1$) | 15800 | 16786 | 3 | 10851 | 10871 | 3 |
| G$k$–TS ($size = 1$) | 15788 | 8397 | 15 | 10889 | 5036 | 15 |
| G$k$–descent ($size = 10$) | 14661 | 12091 | 1 | 10407 | 7918 | 1 |
| G$k$–TS ($size = 10$) | 15091 | 7082 | 10 | 10639 | 4494 | 10 |

Table 2: Results on networks with 200 vertices (total CPU time of 1 hour)

is obtained by setting $iter^*$ to a very large number. Also, compared to the setting used for the diversification strategies, we increase the number of 1-opt evaluations from $pct_1 = 10\%$ to 25% to allow the pure TS to improve its performance. We introduce the greedy 2-opt exchange strategy to better evaluate the impact of: (1) the more "aggressive" 2-opt neighborhood approach, which selects the best move in a fraction of the 2-opt neighborhood, instead of choosing any 2-opt move at random; (2) the variable size $k$ of the $k$-opt move performed at each iteration of the greedy $k$-opt exchange approach. Thus, a total of 17 different heuristics are compared. In addition, the tables display the average value of the best solution found during the initialization step, which uses a greedy SD procedure based on evaluating in priority the shortest path associated to the vertex with the greatest demand value.

We can draw a number of conclusions from these results:

- On most 200-vertex instances, the 1-opt descent performs better than TS, independently of the diversification strategy. This is not the case for the 500-vertex instances (especially with a density of 10%), but even on these instances, TS is only slightly more effective, on average, than descent. We also remark that the pure TS, with no diversification, appears more effective than using TS combined with the diversification strategies AM, GN and 2N. This certainly indicates that there are better parameter settings for these approaches, but more importantly,

| Heuristic | density = 1% | | | density = 10% | | |
|---|---|---|---|---|---|---|
| | $Z(s^{best})$ | $iter_{div}$ | $iter_{loc}$ | $Z(s^{best})$ | $iter_{div}$ | $iter_{loc}$ |
| Initialization | 43492 | – | – | 24310 | – | – |
| TS | 42686 | 1 | 11049 | 23872 | 1 | 6018 |
| AM–descent | 42994 | 101 | 109 | 23928 | 57 | 96 |
| AM–TS | 42835 | 169 | 163 | 23845 | 88 | 157 |
| GM–descent ($size = 1$) | 42944 | 437 | 30 | 23970 | 150 | 35 |
| GM–TS ($size = 1$) | 42857 | 396 | 82 | 23885 | 131 | 101 |
| GM–descent ($size = 10$) | 42929 | 283 | 23 | 23928 | 99 | 30 |
| GM–TS ($size = 10$) | 42858 | 258 | 70 | 23856 | 91 | 88 |
| 2N–descent | 42651 | 384 | 4 | 23724 | 157 | 5 |
| 2N–TS | 42756 | 3489 | 10 | 23910 | 1482 | 10 |
| G2–descent ($size = 1$) | 42670 | 13871 | 1 | 23826 | 6127 | 1 |
| G2–TS ($size = 1$) | 42624 | 3671 | 10 | 23823 | 1578 | 10 |
| G2–descent ($size = 10$) | 42683 | 13794 | 1 | 23842 | 6092 | 1 |
| G2–TS ($size = 10$) | 42630 | 3644 | 10 | 23828 | 1563 | 10 |
| G$k$–descent ($size = 1$) | 42726 | 1995 | 7 | 23736 | 610 | 10 |
| G$k$–TS ($size = 1$) | 42684 | 1262 | 27 | 23694 | 386 | 38 |
| G$k$–descent ($size = 10$) | 42312 | 3093 | 2 | 23606 | 1256 | 2 |
| G$k$–TS ($size = 10$) | 42463 | 1826 | 12 | 23580 | 716 | 13 |

Table 3: Results on networks with 500 vertices (total CPU time of 2.5 hours)

that the 1-opt neighborhood is, on the *long term*, not "rich" enough to identify improving solutions. This observation is corroborated by the values of $iter_{loc}$ observed for the best diversification strategies: they tend towards the minimum number of iterations performed during any single local search phase, which, for TS, is equal to $iter^* = 10$, and for descent, is equal to 1 (no improving move is found at the very first iteration). Thus, the superiority of descent over TS is simply the result of the many useless, longer, local search iterations performed by the latter.

- The starting points provided by the AM approach are not effective enough, as revealed by the relatively large values of the average number of local search iterations, $iter_{loc}$, observed in this case. In fact, the AM approach outperforms only the GM strategy, the sole one that does not use the history of the search process.

- Among all strategies, the GM strategy gives the worst results. It is interesting to note that, by using $size = 10$ instead of $size = 1$, we only marginally improve the performance of the method.

- The behavior of the 2N strategy varies significantly depending whether we use it with descent or with TS. With descent, very few diversification steps are performed, while with TS, the number of construction steps is an order of magnitude greater. The explanation lies in the different fraction of the 1-opt neighborhood explored by each local search procedure, 25% for descent and 10% for TS. This last value should probably be increased to improve the overall performance, especially since 2N-TS performs worse than the pure TS. We also note that the 2N-descent approach is generally more effective than the descent variants ($size = 1$ and $size = 10$) of the G2 strategy, despite the fact that the latter perform an order of magnitude more diversification steps than the former. Since both the 2N and the G2 approaches are based on the same type of move, 2-opt, we conclude that, when using this type of move in a diversification step, it is beneficial to spend more effort on the determination of better starting points through a partial examination of the neighborhood, than to generate more.

- The G$k$ strategy is significantly better than all the others, especially when the pool of elite solutions is used (i.e., when $size = 10$). In particular, we note that the approach outperforms both 2N and G2. This can be easily explained since $k$-opt moves differ more significantly from the 1-opt moves used in the local search phases than the 2-opt moves, and therefore, provide better opportunities for diversification. It is also remarkable that, unlike the GM and G2 approaches, the G$k$ strategy is significantly improved when it uses the pool of elite solutions. Thus, the method is so effective because it combines: (1) a complex type of moves with (2) the generation of several quite different solutions at each construction step, which are then compared with each other, as well as with solutions generated in the past.

# 6 Conclusion

We presented new heuristics for the nonbifurcated network loading problem described in [3]. These heuristics are based on alternating between local search and construction steps. At each construction step, except the initial one, we perform a diversification strategy to provide a new initial point to the local search procedure. We experimentally compared four such strategies, using large-scale instances with up to 500 vertices. The results indicated that the best strategy combines two desirable features of any diversification approach: (1) it generates solutions that are significantly different than the ones obtained through the local search procedure, because it is based on more complex types of moves; (2) it keeps track of the history of the search process by using an adaptive memory based on a pool of elite solutions, filled with the best candidates generated at each diversification step.

# Acknowledgments

Special thanks are due to François Guertin and Serge Bisaillon for their help in implementing and testing our algorithms. Financial support for this work was provided by the Canadian Natural Sciences and Engineering Research Council (NSERC) and by the Quebec Fonds pour la Formation de Chercheurs et l'Aide à la Recherche (FCAR). This support is gratefully acknowledged.

# References

[1] Balakrishnan A., T.L. Magnanti and P. Mirchandani (1998), "Network Design", in *Annotated Bibliographies in Combinatorial Optimization*, M. Dell'Amico, F. Maffioli and S. Martello eds., Wiley, 311–334.

[2] Barahona F. (1996), "Network Design Using Cut Inequalities" *SIAM Journal on Optimization 6*, 823–837.

[3] Berger D., B. Gendron, J.-Y. Potvin, S. Raghavan and P. Soriano (2000), "Tabu Search for A Network Loading Problem with Multiple Facilities", *Journal of Heuristics 6(2)*, 253–267.

[4] Bienstock D. and O. Günlük (1996), "Capacitated Network Design – Polyhedral Structure and Computation" *INFORMS Journal on Computing 8*, 243–259.

[5] Dijkstra E.W. (1959), "A Note on Two Problems in Connection with Graphs", *Numerische Mathematik 1*, 269–271.

[6] Feo T.A. and M.G.C. Resende (1995), "Greedy Randomized Adaptive Search Procedures", *Journal of Global Optimization 6*, 109–133.

[7] Festa P. and M.G.C. Resende (2000), "GRASP: An Annotated Bibliography", Technical report, AT&T Labs Research, Florham Park, NJ.

[8] Gavish B. and I. Neuman (1989), "A System for Routing and Capacity Assignment in Computer Communication Networks", *IEEE Transactions on Communications*, 360–366.

[9] Gavish B. and A. Altinkemer (1990), "Backbone Network Design Tools with Economic Tradeoffs", *ORSA Journal on Computing 2*, 236–252.

[10] Gendron B., T.G. Crainic and A. Frangioni (1998), "Multicommodity Capacitated Network Design", in *Telecommunications Network Planning*, B. Sansó and P. Soriano eds., Kluwer Academic Publishers, 1–19.

[11] Glover F. and M. Laguna (1997), *Tabu Search*, Kluwer Academic Publishers.

[12] Günlük O. (1999), "A Branch-and-Cut Algorithm for Capacitated Network Design Problems", *Mathematical Programming A86*, 17–39.

[13] Hadjiconstatinou E. and Christofides N. (1999), "An Efficient Implementation of an Algorithm for Finding $K$ Shortest Simple Paths", *Networks 29*, 88–101.

[14] Hansen P. and N. Mladenović (1998), "Variable Neighborhood Search: Principles and Applications", Les Cahiers du GERAD G-98-20, Montréal, Canada.

[15] Hansen P., N. Mladenović and D. Perez-Brito (1998), "Variable Neighborhood Decomposition Search", Les Cahiers du GERAD G-98-53, Montréal, Canada.

[16] Hansen P. and N. Mladenović (2000), "Variable Neighborhood Search: A Chapter of Handbook of Applied Optimization", Les Cahiers du GERAD G-2000-03, Montréal, Canada.

[17] Magnanti T.L. and P. Mirchandani (1993), "Shortest Paths, Single Origin-Destination Network Design and Associated Polyhedra", *Networks 23*, 103–121.

[18] Magnanti T.L., P. Mirchandani and R. Vachani (1993), "The Convex Hull of Two Core Capacitated Network Design Problems", *Mathematical Programming 60*, 233–250.

[19] Magnanti T.L., P. Mirchandani and R. Vachani (1995), "Modeling and Solving the Two-Facility Capacitated Network Loading Problem" *Operations Research 43*, 142–157.

[20] Magnanti T.L. and R.T. Wong (1984), "Network Design and Transportation Planning: Models and Algorithms", *Transportation Science 18*, 1–55.

[21] Minoux M. (1989), "Network Synthesis and Optimum Network Design Problems: Models, Solution Methods and Applications", *Networks 19*, 313–360.

[22] Ng T. and D. Hoang (1987), "Joint Optimization of Capacity and Flow Assignment in a Packet-Switched Communication Network", *IEEE Transactions on Computing 35*, 202–209.

[23] Resende M.G.C. (2000), "Greedy Randomized Adaptive Search Procedures (GRASP)", *Encyclopedia of Optimization*, Kluwer Academic Publishers.

[24] Rochat Y. and É.D. Taillard (1995), "Probabilistic Diversification and Intensification in Local Search for Vehicle Routing", *Journal of Heuristics 1*, 147–167.

[25] Shier D.R. (1979), "On Algorithms for Finding the k-Shortest Paths in a Network", *Networks 9*, 195–214.

[26] Taillard É.D., L.M. Gambardella, M. Gendreau and J.-Y. Potvin (2000), "Adaptive Memory Programming: A Unified View of Metaheuristics", forthcoming *European Journal of Operational Research.*

# Appendix A

In this Appendix, we describe how we perform a random selection of one element in a list of elements ordered from best to worst. The best element is given the value *Max*, while the worst element is given the value *Min*. The values for the other elements are equally spaced between *Min* and *Max*. More precisely, assuming $d$ elements in the list (with $d > 1$), the value $v_i$ of an element at rank $i$ is computed as:

$$v_i = Max - (Max - Min) \times \frac{i-1}{d-1}, \quad 1 \leq i \leq d.$$

The selection probability $p_i$ of the element at rank $i$ is then:

$$p_i = \frac{v_i}{\sum_{j=1}^{d} v_j}, \quad 1 \leq i \leq d.$$

By imposing that $Min + Max = 2$, the selection bias in favor of the best elements can be increased by setting the *Max* value closer to 2, or reduced by setting its value closer to 1. In our implementations, we use the values $Min = 0.5$ and $Max = 1.5$.